N88-19120

Toward Automatic Finite Element Analysis

Ajay Kela
Research Assistant
Renato Perucchio
Assistant Professor
Production Automation Project
Mechanical Engineering Department
University of Rochester
Rochester, N.Y.

Herbert Voelcker
Professor
Computer Aided Design Project
Sibley School of Mechanical
and Aerospace Engineering
Cornell University
Ithaca, N.Y.

Two problems must be solved if the finite element method is to become a reliable and affordable "blackbox" engineering tool. FE meshes must be generated automatically from CAD databases, and mesh analysis must be made self-adaptive. The experimental system described here solves both problems in 2-D through spatial and analytical substructuring techniques that are now being extended into 3-D.

The essence of mechanical design is interplay between human creativity and incisive analysis. The procedure for designing a critical component or structure typically runs as:

1. Prepare a candidate design.
2. Analyze the design using the finite element (FE) method.
   (a) Model the designed structure and its loading and constraints.
   (b) Analyze the loaded model.
   (c) Assess the validity of the analytical results.
   (d) Repeat steps 2(a—c) until acceptable analytical results are obtained.
3. Assess the candidate design.
4. Repeat steps 1—3 until the design is acceptable.

Thus the design process is doubly iterative because current FE techniques are not single-shot blackbox tools with guaranteed reliability; they require human judgement and "tuning." It follows that the (in)efficiency of the inner analysis loop is a strong determinant of the quality of the final design when the cost of design matters, as is usually the case. If analysis can be made cheap, fast, and reliable, more alternatives can be considered and better designs will result.

Let's look more closely at the analysis procedure. During step 2(a), the design is modeled as a properly connected mesh of suitably sized and shaped elements (triangles, quads, etc.) from an element library. Its loading and constraints are modeled by assigning suitable constants (e.g. displacement and load values) to particular nodes of the mesh. The operative words here are "suitably sized and shaped" and "properly connected". If the elements are too large or have bad aspect ratios, or if the mesh as a whole does not obey the combinatorial sharing rules of FE mesh decompositions, inaccurate and inconsistent results will accrue because the mathematical conditions underlying the FE method will have been violated. In the early days of FE analysis, the analyst was wholly responsible for mesh and element integrity. Today, computer graphics preprocessors help ensure proper connectivity, but the selection, placement, and sizing of elements are still the user's responsibilities.

Step 2(b), analysis of the loaded model, is usually performed by using a standard code such as Nastran and Ansys. This step is largely automatic, and the popular codes are well debugged though sometimes expensive to run.

For step 2(c), assessing the validity of the results, there are no standard methods and the analyst's judgement plays a critical role. In the early days, when "results" were huge tables of numbers, assessment was largely a black art. Graphics postprocessors, which can display colored contour plots of stresses, temperatures, and so forth, enable experi-

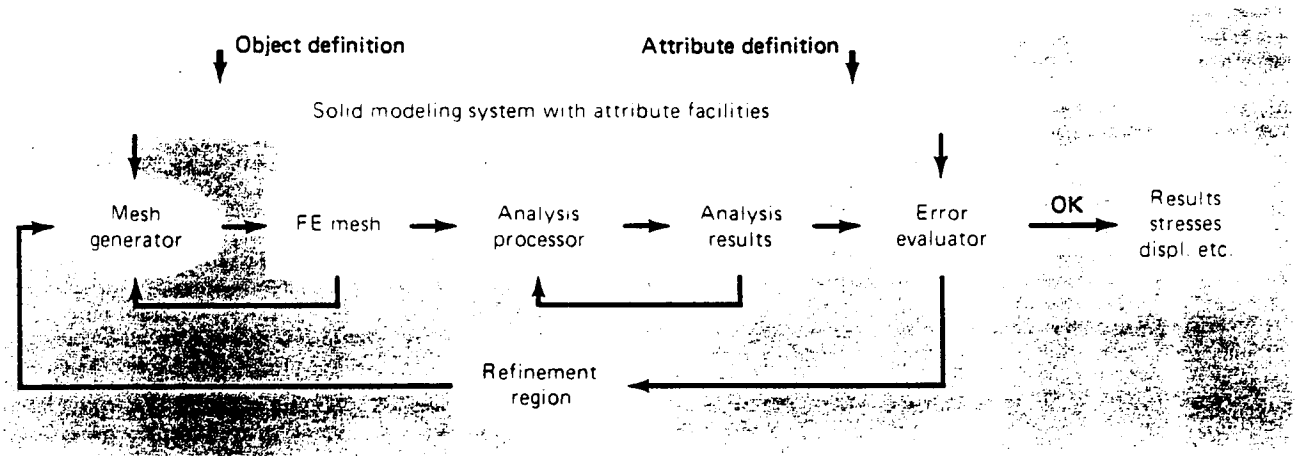


Fig. 1 An automatic finite element analysis system.

enced analysts to identify trouble spots (such as regions with high cross-element gradients) quite effectively.

During step 2(d), the analyst refines the mesh by subdividing troublesome regions into smaller elements, and then reanalyzing the whole.

Obviously, automation of the whole process will make design more systematic and efficient by replacing the analyst's judgement with mathematical criteria. Two new tools make automation of the FE mesh feasible:

- Solid modeling technology [1, 2] enables designers to create and store in CAD systems informationally complete "master models" of mechanical parts and products. From there, one should be able to generate FE meshes automatically.
- New algorithms for analyzing errors in a finite element analysis [3—7] systematic means to automate the results assessments of step 2(c).

One more tool is needed: a good method for using error indicators to refine the FE mesh automatically. Another tool, while not essential, is also very desirable: a method for analyzing refined meshes selectively or incrementally so that results already computed for unmodified regions of a mesh can be reused rather than recomputed.

Figure 1 shows a design for an automatic analysis system. In this system, the user defines the structure to be analyzed in the Solid Modeling System (SMS) together with attributes such as boundary conditions, loads, material properties, and certain analytical parameters. The mesh generator produces a discretized model (the FE mesh) from the geometric definition and attribute specifications. (Attributes can determine, for example, the positions of some nodes.) The analysis processor performs FE analysis: it computes primary and secondary field variables (in general, the displacements vector at nodal points and the stress tensor within the elements) for the loaded and constrained FE mesh. Finally, the error evaluator compares error estimates derived from the analysis output with specified tolerances, and either accepts the results or requests a new analysis of a modified mesh. In the latter case, the error evaluator indicates the regions in the current model that require refinement. The inner mesh-generation loop and mesh-analysis loop in Figure 1 connote localized mesh refinement and incremental reanalysis.

This approach to automatic FE analysis has been embodied in an experimental 2-D system whose underlying princi-

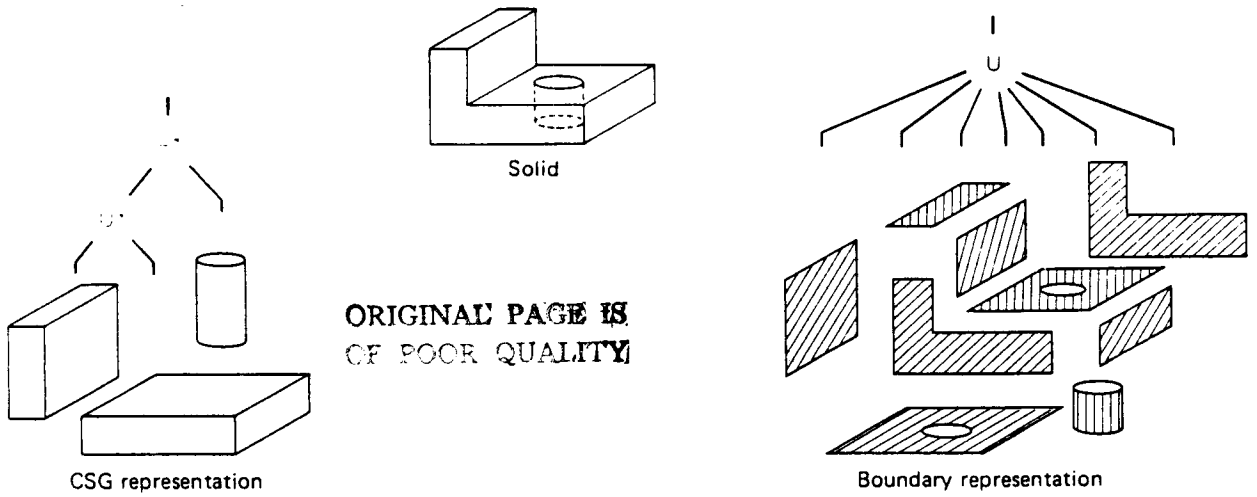


Fig. 2 Two unambiguous representation schemes for solids.

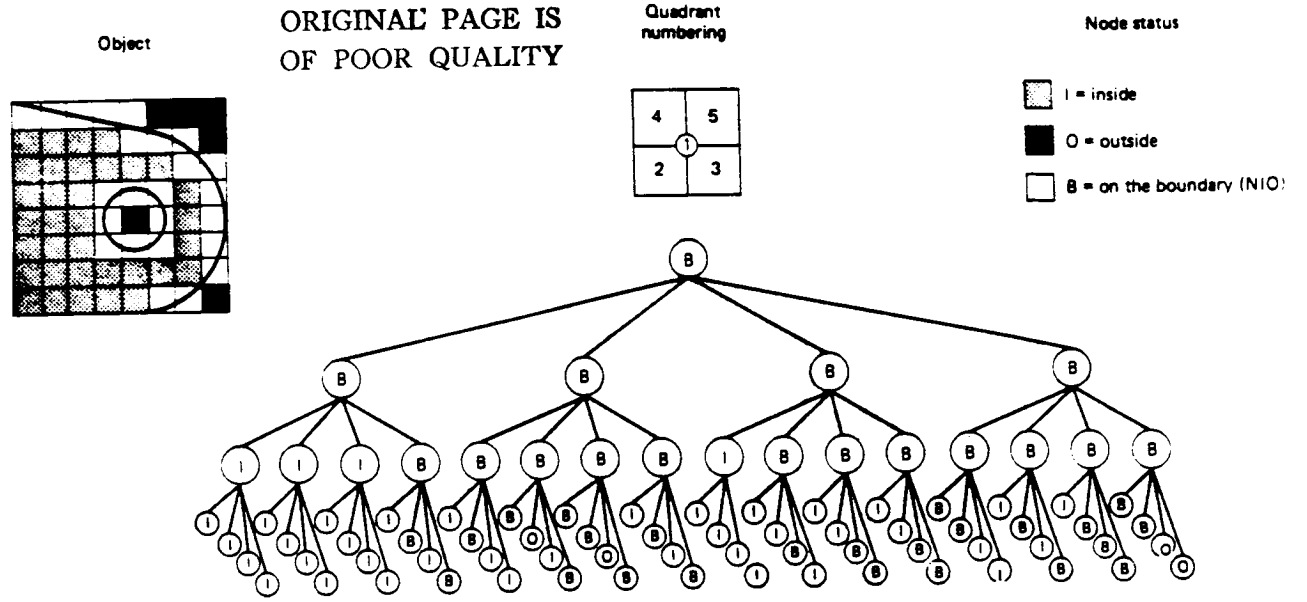


Fig. 3 A quadtree approximation.

ples will be explained. (Our actual implementation is somewhat different than Figure 1 for reasons of computational efficiency.) All meshes and analytical results that appear in this article were produced with this experimental system. This article summarizes a moderately complicated topic: for technical details, see [8].

Automatic Mesh Generation

Most "automatic" meshing utilities in contemporary CAD systems actually operate from wireframe descriptions of objects via mapping algorithms. The user must partition the domain, which is represented by a collection of edges, into a set of topologically simple subdomains in which meshes can be generated automatically. This approach is unsuitable for a fully automatic meshing procedure because it depends on human judgement both to guide meshing and to resolve ambiguities in the wireframe representation.

Genuinely automatic mesh generation must start from an unambiguous representation of the object to be analyzed, and thus needs some form of SMS. Nearly all current SMS systems are based internally on one or both of the representation schemes illustrated in Figure 2 [1, 2]. Constructive Solid Geometry (CSG) exploits the notion of "adding" and "subtracting" simple solid building blocks (via set-union and set-difference operations). Boundary schemes describe solids indirectly via sets of faces which are represented by sets of edges that bound finite regions of surfaces. The various schemes that have been proposed for automatic mesh generation can be divided into two families: recursive spatial subdivision (quadtree and octree) schemes, and triangulation and other schemes. After a brief discussion of the second family, we will focus on the first.

Triangulation and Other Schemes

Wordenweber [9] and Cavendish [10] have developed two different two-stage approaches to automatic triangulation of solid domains. Wordenweber's procedure first does surface triangulation of the boundary of the solid, and then performs solid triangulation in the interior. The tetrahedral meshes that result are coarse and usually contain distorted elements that must be refined to be useful for analysis.

In the Cavendish method, points are injected into the solid, and then a solid triangulation is induced in which the points become nodes of tetrahedral elements. The main working tool of the second-stage triangulation is a Delaunay algorithm that generates valid meshes of tetrahedral elements within convex hulls of node points. Good methods are still being sought for inserting points automatically during the procedure's first stage.

In both of these approaches, mesh refinement is done by splitting existing elements. Because refinement is driven from an FE mesh rather than from the original solid model, refinement does not improve the geometric approximation of the original solid. Also, the meshes are not spatially addressable.

A few commercial CAD systems claim automatic meshing facilities that can involve triangulation but the principles are proprietary. Lee's method [11], which has been described publicly and implemented in 2-D, exploits the decomposition inherent in CSG representations rather than triangulation or spatial subdivision. Briefly, Lee generates "natural" distributions of points in each CSG primitive and then induces a uniform spatial distribution of points over the whole object by "thinning" points in regions where primitives overlap; a mesh of quadrilateral and triangular elements is then grown over the points in the object.

Recursive Spatial Subdivision

We approximate the object to be meshed with a union of disjoint, variably sized rectangles (in 2-D) or blocks (in 3-D). These are generated by recursively subdividing a spatial region enclosing the object, rather than the object itself. Figure 3 shows a 2-D example.

The object (a rounded plate with a hole) is "boxed" to

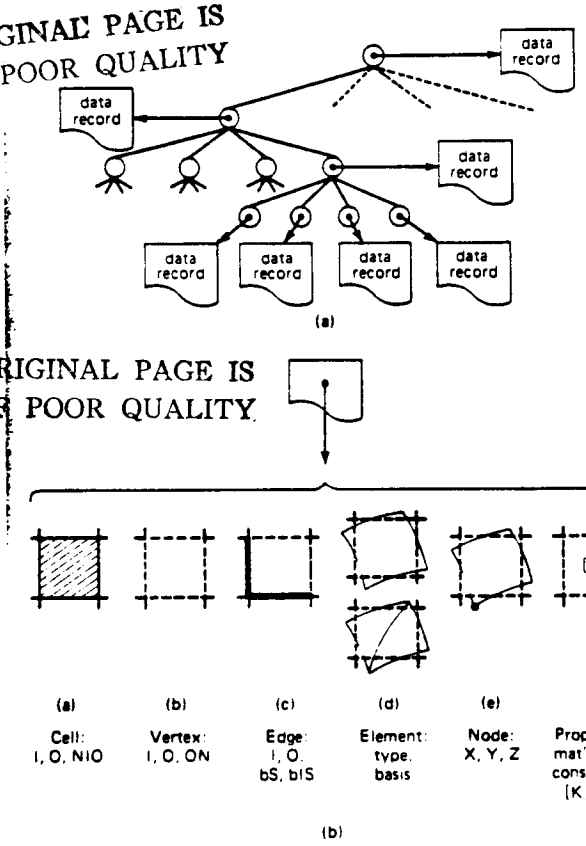


Fig. 4 Hierarchical structure for the FE model.

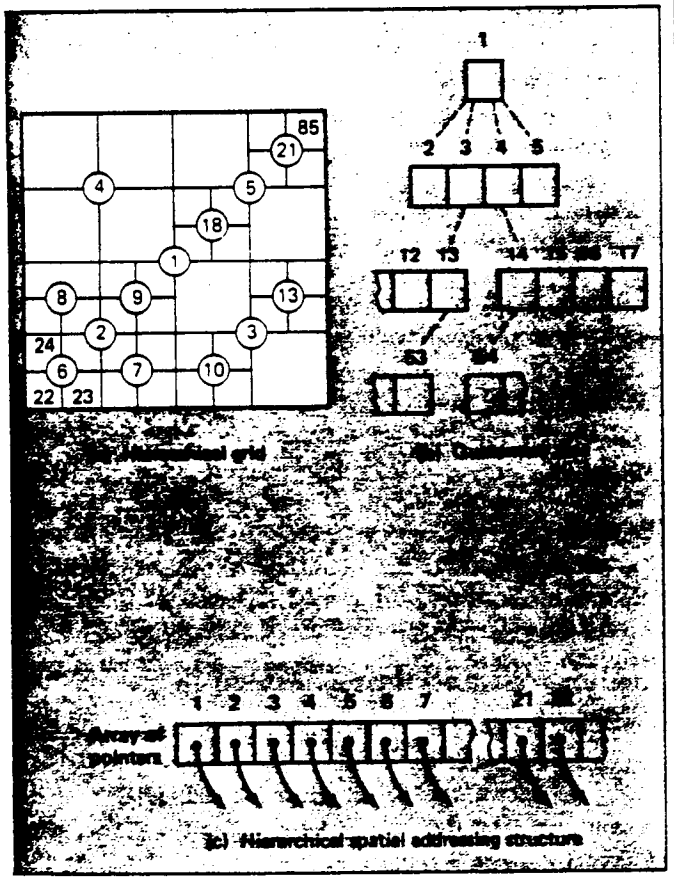


Fig. 5 Directly addressable hierarchical grid.

establish a convenient minimal spatial region, and then the box is decomposed into quadrants. When a quadrant can be classified as wholly inside or outside of the object, subdivision ceases; when a quadrant cannot be so classified, it is subdivided into quadrants. So this process continues until some minimal resolution level is reached. (In 3-D, the decomposition proceeds by octants.) Approximations produced this way can be represented by logical trees whose nodes have four or eight sons (see Figure 3), hence the popular names "quadtree" and "octree" [12].

As we will explain, inside cells of a spatial decomposition can be easily converted into "nice" mesh elements, but boundary cells require further processing lest their literal translations into mesh elements introduce bogus high-gradient stress regions in the analytical results. We'll deal with boundary-cell processing later; for the moment, assume that the "B" cells in Figure 3 are somehow reshaped into valid mesh elements that closely approximate the object's boundary.

Recursive spatial decompositions have two intrinsic properties, hierarchical structure and spatial addressability, that are central to the mesh refinement and incremental analysis techniques described later. These intrinsic properties, plus an extrinsic (engineered) property called logical addressability, warrant discussion.

Hierarchical structure. The tree structure in Figure 3 results from the subdivision rule used to produce the decomposition, and one can think of the tree as an organizing or cataloging structure for data describing particular regions of space.

Figure 4(a) illustrates this notion by showing a data record associated with each node of the tree; Figure 4(b) shows data pertinent to automatic mesh generation that might be stored within such a record. These include classification of the spatial region represented by the node as inside, outside, or on the boundary (Figure 3); shape functions for a few (typically one) finite elements associated with the region; and properties associated with the finite elements, such as one or more stiffness matrices, external constraints, and so forth.

At the lowest level of the tree one finds the smallest spatial regions and simplest finite elements. As one ascends the tree the regions become larger (encompassing multiples of four or eight elemental regions) and the finite elements become superelements with associated ("assembled") stiffness matrices, collected constraints, and so forth. Such an organization is ideally suited to mesh refinement by subdivision and incremental mesh analysis.

Logical addressability. Given the notion of a tree as an organizing structure for hierarchical spatial data, how should such a structure be mapped into computer storage as a data structure, and how does one gain access to it to store and retrieve data? The tree diagrams in Figures 3 and 4 suggest the classical approach: represent a tree with a linked list in which nodes are addressed indirectly through downward pointers to sons and perhaps lateral pointers to siblings. The data record associated with each node is addressed through a

special pointer stored with the node. Thus one has access to data by following pointers downward from the root of the tree.

Alternatively, a recursive spatial decomposition can be viewed as a directly addressable hierarchical grid (see Figure 5) in which the number of cells in each linear dimension is an integer power of two. The key here is a systematic scheme for numbering all possible nodes of the underlying tree. In Figure 5(a), "1" represents the enclosing box, 2—5 represent specific quadrants of "1," "6"—"9" represent quadrants of "2," and so on. The underlying relation, which can be applied recursively, is:

The four sons of a parent node P are [4 * P − 2, 4 * P − 1, 4 * P, 4 * P + 1], and the parent of P is (P + 2) div 4.

These numbers can be used as indices for a single array of pointers to data records, as shown in Figure 5(c). Thus, to access the spatial data for a particular node in the underlying tree, one merely calculates an array index through a simple formula and follows the single pointer stored there. This is usually much faster than the pointer-following method noted above but it carries a storage penalty. Specifically, the pointer array in Figure 5 (c) must be large enough to accommodate all possible nodes in the tree.

If the lowest-level grid in Figure 5 (a) requires $N*N*K$ units of storage ($N*N*N*K$ in 3-D) for pointers and data records, one needs:

$$\frac{K * (2^{D * (1 + \log N)} - 1)}{2^D - 1}$$

units of storage for the worst-case whole tree, where D is the dimension of the space and "log" is log-2. Thus a 2-D hierarchical grid requires at most about 33 percent more storage than the $N*N*K$ units needed for its lowest level: in 3-D only about 14 percent more storage is needed.

Spatial addressability. Suppose that we know the geometric size and spatial position of the "1" cell (the overall box) in Figure 5(a). We can quickly compute the index of any cell in the hierarchy from its size and position, and conversely from an index we can quickly compute the size and position of the associated spatial cell (an example is in Table I). We have already seen that cell indices allow access through a single pointer to data associated with the cell, and thus we can associate, without searching, spatial regions with stored data and stored data with spatial regions. This is what is meant by spatial addressability.

In practical terms, if a particular region of an object proves troublesome either in mesh generation or mesh analysis, one has direct access to pertinent mesh and analytical data to take localized corrective measures.

An Automatic Meshing Procedure Based On Spatial Subdivision

This procedure produces a spatially addressable FE mesh embedded in the lowest level of a hierarchical grid. Higher levels of the grid are used during construction of the mesh and when the mesh is analyzed, refined, and incrementally reanalyzed. The procedure starts with a representation in an SMS of the object to be meshed, and operates in two stages. The first stage meshes the interior of the object by spatial subdivision and the second extends the mesh to the object's boundary. The following descriptions are in 2-D; 3-D extensions are in the final section.

The use of quadtree and octree methods for automatic mesh generation was pioneered by Shephard and Yerry [13, 14]. Our work is similar to theirs but important differences will be noted as we go along.

Stage 1: interior meshing. The object S, Figure 6(a), is enclosed in a box, Figure 6(b), which is recursively subdivided into a grid whose smallest cell size determines the element size (or element density) of the initial FE mesh. This minimal size is determined by subdividing cells until no cell contains more than one connected boundary segment of S. As the subdivision proceeds the cells are classified as being "IN" S, "OUT" of S, or neither in nor out ("NIO"). Cells

Table I

Finding a Spatial Position from a Cell Index

This procedure computes the center position and half-size of cell P when the enclosing box (P=1) is square, has size (*boxsz*) and is centered at (*bx, by*). The procedure is invoked as:

GetPosition (P, cellx, celly, cellsz)

where *cellx, celly* are the *x,y* coordinates of the center of the cell and *cellsz* is the half-size of the cell; *cellx, celly*, and *cellsz* are initialized respectively as *bx, by*, and *boxsz*/2.0. The following algorithm is based on the cell numbering scheme shown in Figure 5.

```
procedure GetPosition (P, cellx, celly, cellsz);

{Traverse tree upwards to root-node, from cell (P)
 marking all ancestors of the cell.}

CellNo:=P;
ParentLevel:=1;
while (CellNo<> 1)
do begin
  SonNo[ParentLevel]:= (CellNo+2) rem 4;
  CellNo:= (CellNo + 2) div 4;
  ParentLevel:= ParentLevel + 1;
end; { while }

{Move down the tree through the ancestors of the cell
 updating each ancestor location to terminate at cell
 location.}

for I:=ParentLevel-1 downto 1
do begin
  cellsz:=cellsz/2.0;
  case SonNo[I]
  of
    0: cellx:=cellx-cellz; celly:=celly-cellz
    1: cellx:=cellx+cellz; celly:=celly-cellz
    2: cellx:=cellx-cellz; celly:=celly+cellz
    3: cellx:=cellx+cellz; celly:=celly+cellz
  end; { case }
end; {for }
end; { GetPosition }
```

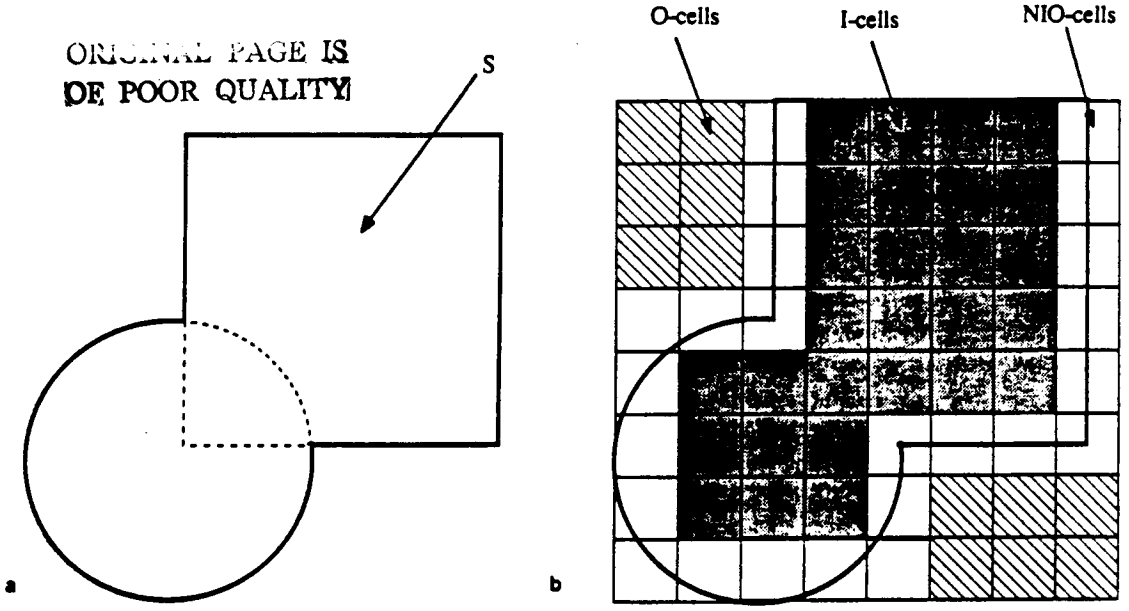


Fig. 6 First stage of the automatic meshing algorithm.

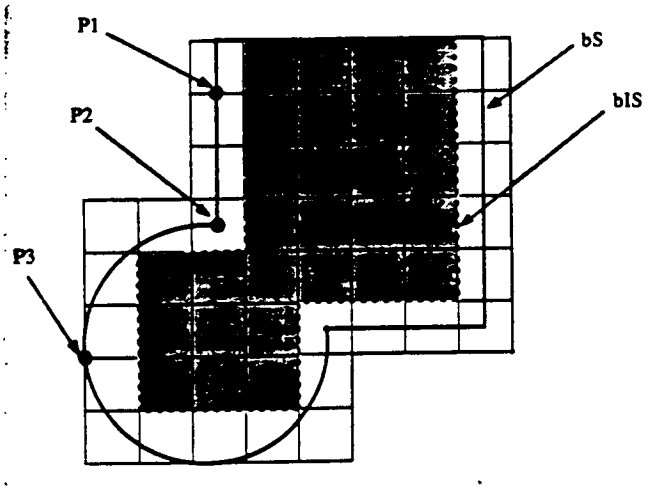


Fig. 7 Generation of *bS* nodes in stage 2 of the meshing algorithm.

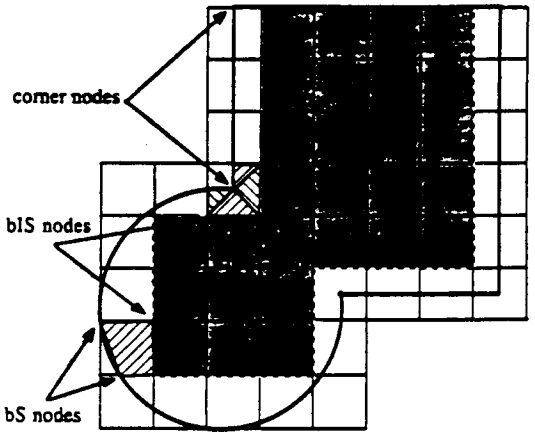


Fig. 8 Element generation via linking *bS* and *bIS* nodes.

classified as IN at higher levels in the hierarchy are subdivided to the final grid size without further classification. The collection of IN cells constitutes the interior mesh of S.

The main computational utility used for cell classification is the modified cell classification procedure:

ModClassCell(cell, solid) = ("IN", "OUT", "?")

which is described in [15].

ModClassCell tests a cell to determine if it is entirely inside the solid, entirely outside, or undetermined. The "?" cells are further subdivided and tested. Stage 1 ends with special operations that reclassify final-sized "?" cells as IN, OUT, or NIO. (Some might think that "?" cells must always be NIO, but this is not true for Lee's efficient use of the classification procedure, which assumes a CSG representation of the solid S [15]. Although CSG implementations can be designed to insure that "?" cells are NIO, and the procedure can be used for solids represented in boundary format, both approaches are computationally expensive.)

Specifically, the vertices of each final "?" cell are classified; if one to three vertices are OUT, the cell is NIO. In cases where all four vertices have the same classification the cell is classified as:

if ($Cell \cap^* S = 0$) then "OUT"
else if ($Cell \cap^* S = Cell$) then "IN"
else "NIO"

where $\cap^*$ is the regularized intersection operator [16]. Methods for performing the tests above are described in [8].

We note that the Shephard-Yerry cell classification procedure [13, 14] is based on in/out tests of cell vertices, with some special operations performed on vertices of cells having uniform vertex classifications. In/out tests on vertices are insufficient because cells containing holes or thin sections might be misclassified.

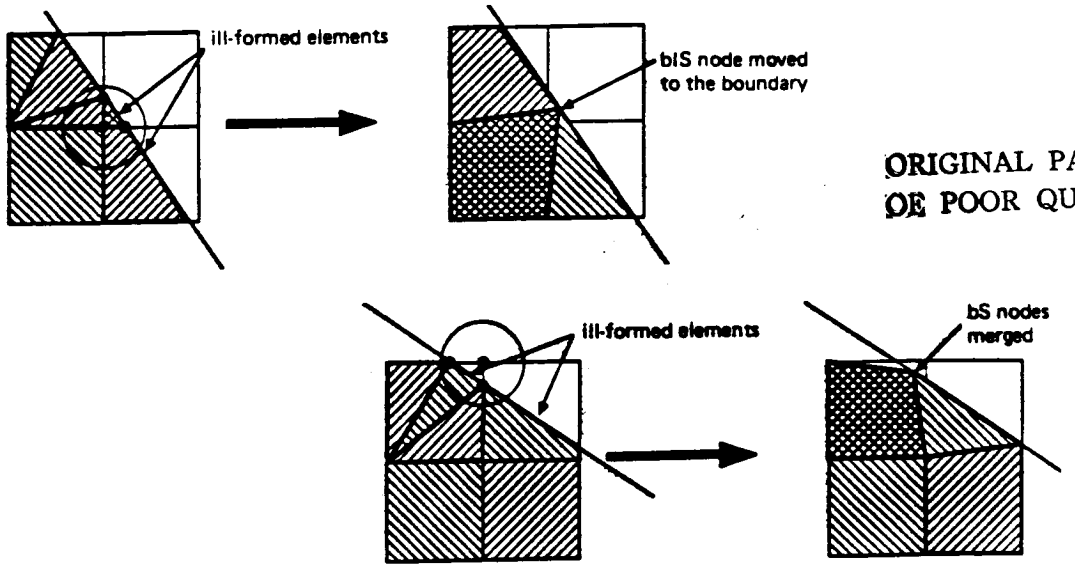
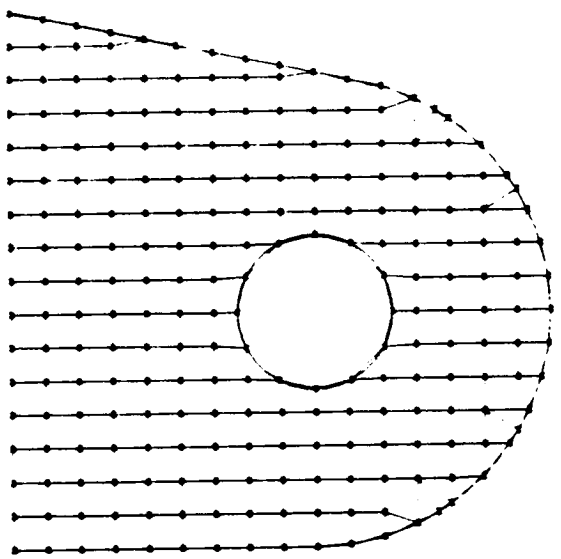


Fig. 9 Node relocation to get well-formed elements.



Stage 2: boundary-region meshing. The task here is to fill the region between the boundary of the interior mesh (denoted *bIS* in Figure 7) and the boundary *bS* of the solid *S*. Observe that:

$$bS \subset (\cup \text{ "NIO" } cells) \cup bIS$$

Thus *bS* is usually contained in the NIO cells and special element-building operations are required, but sometimes segments of *bS* coincide with *bIS*, as at the top of Figure 6(b), and no special processing is needed. We can mesh the interboundary region by visiting each NIO cell and creating elements that link the *bS* segment passing through it to the interior of the solid.

There are three main issues in this process: to devise a systematic way to insure that all NIO cells are visited, to create nodes on *bS*, and to associate *bS* nodes with existing *bIS* nodes to form valid elements.

All NIO cells can be visited by an exhaustive scan of the

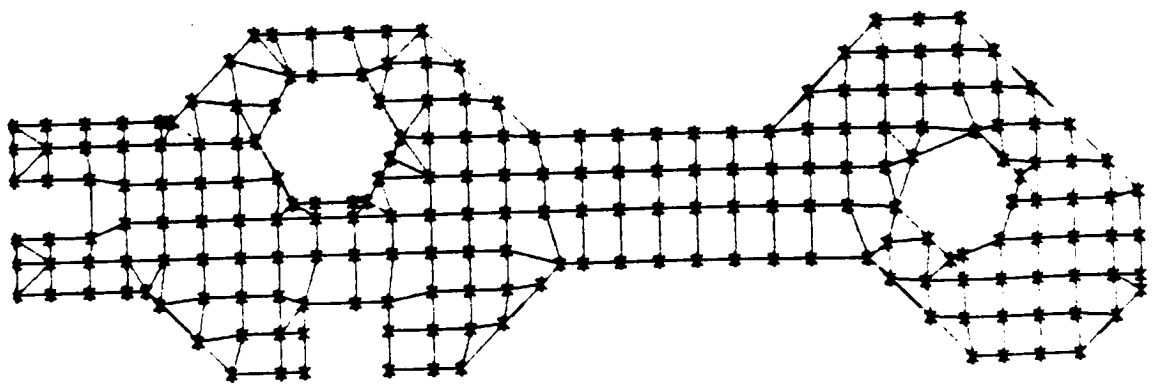


Fig. 10 Examples of automatically generated FE meshes.

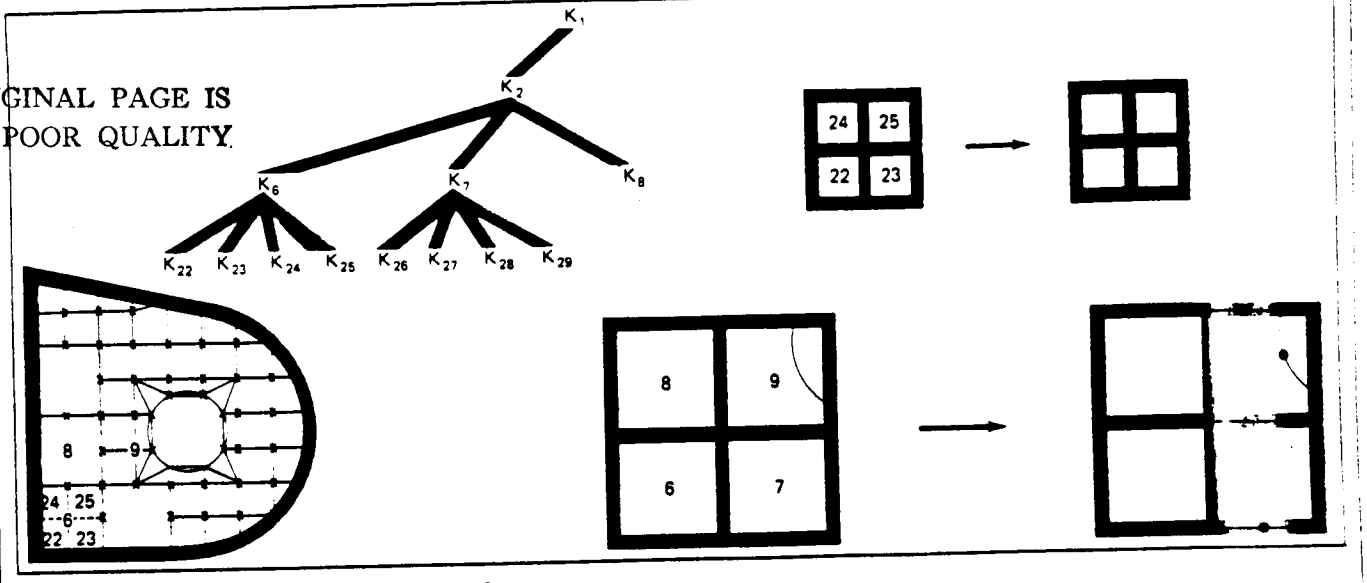


Fig. 11 Assembly via multi-level substructuring.

lowest-level grid, or by tree traversal, or by traversal of *bS*. Since no single approach seems to offer substantial advantages we use grid-scan for generating the initial mesh and, because operations tend to be more localized, tree-traversal for remeshing and reanalysis.

Figure 7 shows *bS* nodes P1, P2, P3 that are created in the following manner. Vertices of *bS* within each NIO cell (e.g. P2 in Figure 7) are tagged as such and are always used as finite element nodes. The vertices of bS are available explicitly if *S* is represented in boundary format. If only a CSG representation is available, as in our system, a limited form of boundary evaluation [17] must be performed. In 2-D, the CSG primitives that intersect an NIO cell are themselves intersected to generate candidate *bS* vertices; the candidates are then classified to identify true *bS* vertices. The analogous 3-D operations amount to constructing a wireframe representation from a CSG representation. Additional *bS* nodes are created by intersecting *bS* with the boundaries of the NIO cells (P1 and P3 in Figure 7).

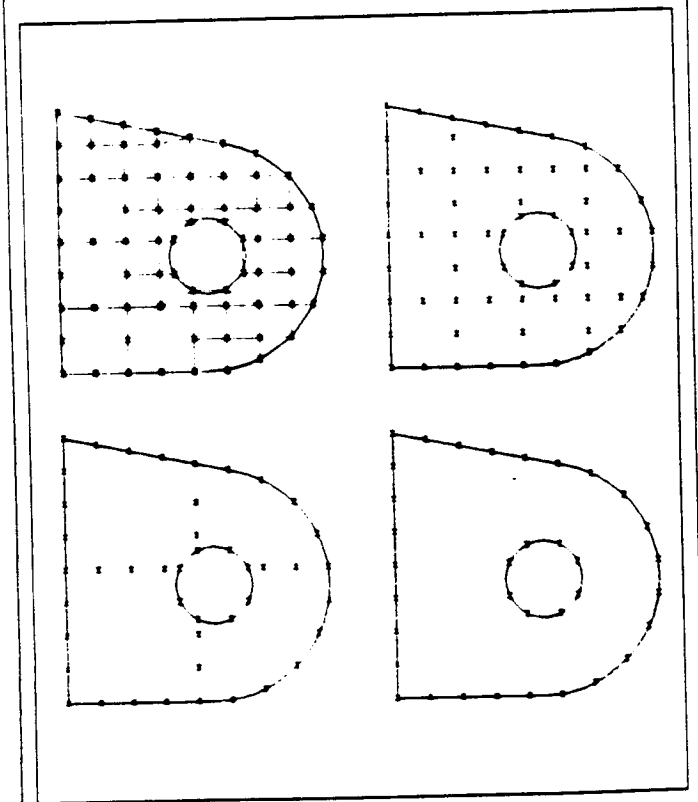


Fig. 12 Substructures at various levels during assembly.

The generation of valid elements within an NIO cell is straightforward if the cell does not contain *bS* vertices (corner nodes): nodes on *bS* and *bIS* belonging to the same NIO cell are simply linked to form quadrilateral and triangular elements (see the lower left portion of Figure 8). The treatment is more involved when a corner is present; a detailed explanation is in [8]. Briefly, the corner node is linked to *bS* and *bIS* nodes within the cell to form a web of triangular elements (Figure 8). To avoid generating elements with poor aspect ratios, the distances between nodes are checked by using a node neighborhood test, and closely spaced nodes are merged into single nodes on *bS*. Figure 9 provides two examples of this process.

The FE mesh is complete at the end of stage 2 of the design procedure. A regular mesh of quadrilateral elements in the interior results from a direct mapping of IN cells. On the boundary, NIO cells are associated with quadrilateral and triangular elements. It is important to note that the FE mesh inherits the spatial addressability and structure of the hierarchical grid because elements and substructures are associated with the quadrants of the original decomposition. Figure 10 shows two examples of meshes generated by our automatic procedure.

The Shephard-Yerry (SY) boundary region meshing algorithm performs in/out tests on the midpoints and quarter-points of the edges of NIO cells, and then maps each NIO cell into one of a finite number of cut-quadrant forms; each cut quadrant is then meshed. (We avoid such geometric approximations by computing exact points of intersection on *bS*.) The final stages of the SY algorithm move nodes in NIO cells to the boundary, and then eliminate ill-formed elements by using a Lagrangian relaxation procedure to smooth a triangulated version of the entire mesh. This last operation

destroys the uniform quadrilateral interior mesh and also spatial addressability, because elements are not constrained to remain in their original cells.

Analysis Of Hierarchical Meshes

We will now summarize a mesh-analysis procedure that exploits the properties of the hierarchical, spatially addressable meshes already described. Recall that data specifying the finite elements in the initial mesh are accessed through the lowest level of the hierarchical grid; Figure 4(b) shows the types of data that are carried.

One analytical simplification is immediately obvious: because the interior mesh elements are uniform, their stiffness matrices are identical if the material properties are homogeneous and thus only one stiffness matrix need be computed for all of the interior elements. Other more important analytical simplifications accrue during both assembly and solution of the system of equations because the hierarchical grid, which so far has provided spatial substructuring for meshing, can serve also as a multilevel analytical substructuring mechanism.

Assembly procedure. Most FE analysis procedures build a single stiffness matrix to cover the whole domain. Our assembler builds and stores stiffness matrices for every non-OUT cell in the hierarchical grid. This is done from the bottom up (see Figure 11) by assembling son matrices and "condensing out" interior degrees-of-freedom to build parent matrices at each level. The parent nodes of the interior mesh with identical (uniform) sons to yield identical substructures and need be assembled only once. The mesh generator tags identical interior-mesh nodes at all levels of the tree to allow this.

Figure 12 shows an initial mesh and substructures at various levels in the assembly process. Note in Figure 12(a) that the initial mesh contains some higher-level substructures; these arise not from assembling lowest-level IN elements, but from intermediate-level cells that were classified as IN and tagged as substructures during stage 1 meshing. (The identical stiffness matrices for lowest-level IN cells are needed in the assembly process only when IN elements must be assembled with elements in NIO cells.)

Solution. Figure 13 illustrates various stages in the solution process. After loads and boundary conditions are attached to the root structure, the FE solver computes the displacements of all nodal points on the boundary, i.e., the nodal points of the root substructure as in Figure 13(a), and then traverses down the tree, recovering displacements of substructure nodes at each level.

The displacements at all levels are saved in data records accessed through the hierarchical grid, and the lowest-level displacements are used to compute the stresses in the elements. Figure 14 shows the displacements and average value per element of a stress component. The displacements in Figure 15 are exaggerated for clarity. All analyses here are linear-static, based on linear isoparametric elements. For nonlinear analysis, where displacements can be large, spatial addressability is still maintained via a backward mapping that associates each displaced element to the original grid.

Remarks

Our experience with this substructuring approach to analysis leads to some conclusions. The hierarchical grid used for mesh generation has almost all of the data management facilities needed for analytical substructuring. The computing time and storage requirements for internal-element assembly are substantially reduced. We have not yet compared the solution efficiency of our tree-traversal method with that

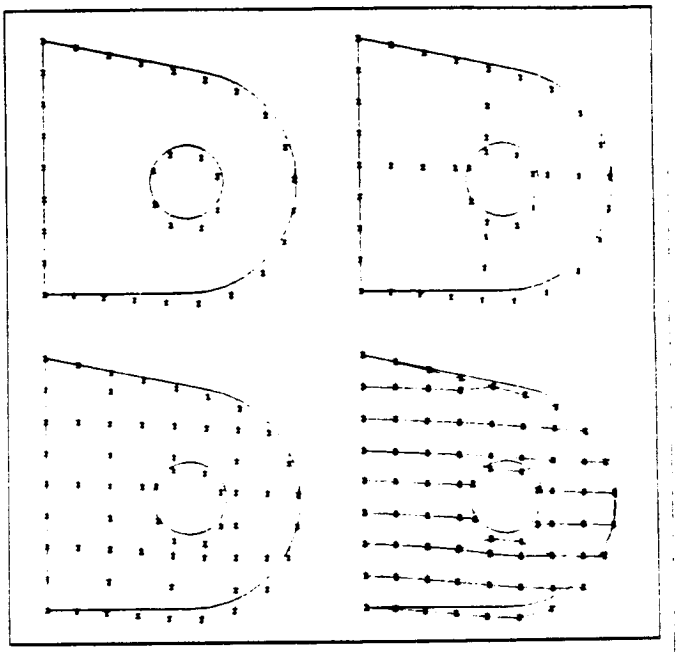


Fig. 13 Nodal displacements at stages of the solution process.

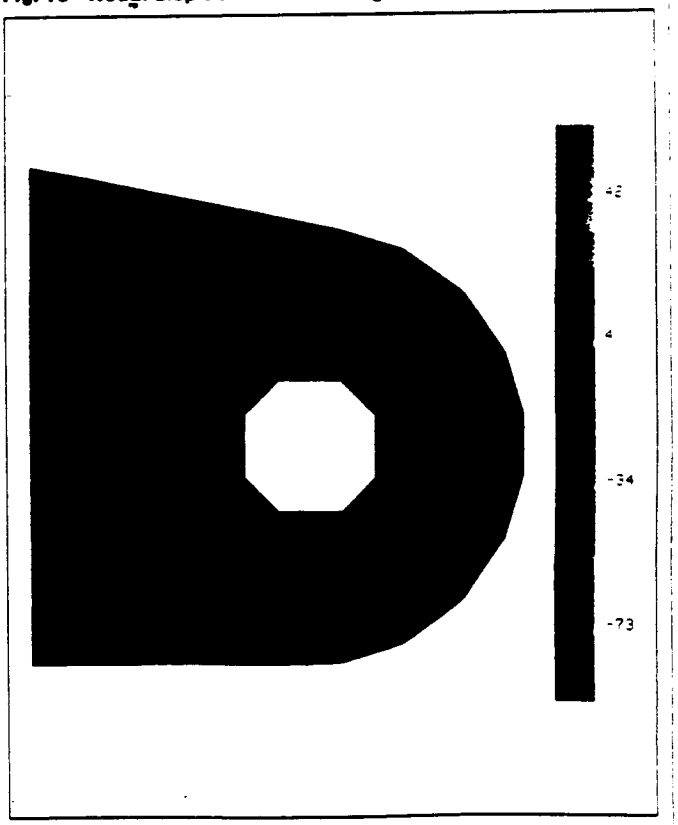


Fig. 14 Average value per element of a stress component.

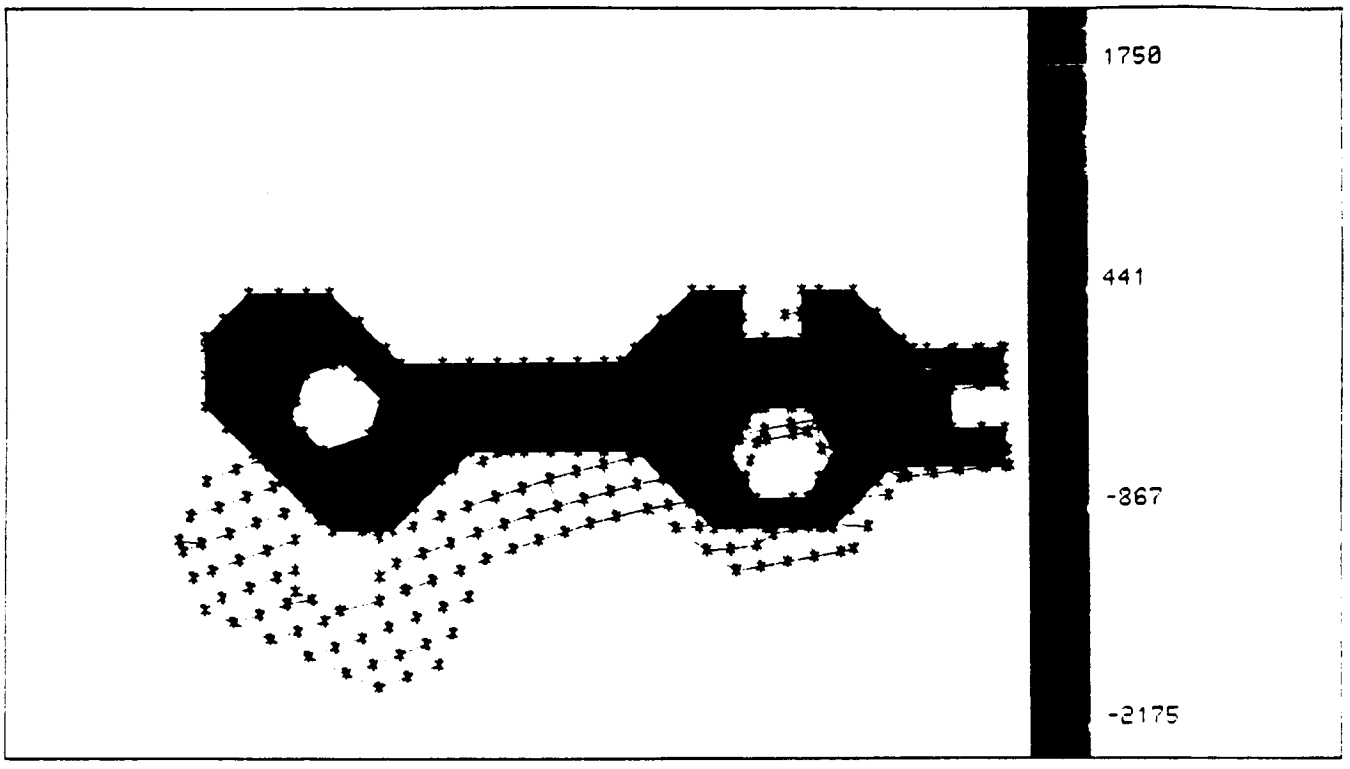


Fig. 15 A bicycle spanner in action. The four nodes on the right-side notch are totally constrained to model engagement with a nut. The average value of a stress component is also shown

of standard solvers, in part because we have made no effort to optimize our code. However, the incremental reanalysis facilities described later clearly outclass standard solvers when it comes to adaptive analysis. Note that solution via tree traversal does not require the normally expensive global element- or node-numbering schemes used by standard solvers to minimize bandwidth or wavefront. Finally, substructuring based on trees lends itself naturally to parallel processing.

In general, substructuring has proven to be efficient [18] and our particular approach to substructuring seems promising for nonlinear as well as linear analysis. In many practical problems (e.g. contact problems, fracture mechanics, and localized plasticity), nonlinear behavior occurs in isolated regions, and spatially localized analytical methods should prove to be efficient. For example, during analysis, regions that become nonlinear can be tagged in the grid and specially handled. In other types of problems one might want displacements and stresses only in small critical regions, and again spatially localized methods seem very appropriate.

Self-Adaptive Incremental Analysis

Assume that a mesh has been constructed at the lowest level of the grid; the mesh has been analyzed and the results stored in the grid (e.g. "f" in Figure 4); and evaluation of the results (discussed next) has indicated that refinement is needed in a particular spatial region, say that represented by the mesh fragment in Figure 16(a).

Two avenues for refinement are available, h-refinement and p-refinement. In p-refinement, illustrated in Figure 16(b), successively higher-order shape functions are assigned to the element formulation. To refine a particular element, the old stiffness matrix for the element is invalidated and a new matrix is computed from the new shape function. No new tree nodes are generated, but the size of the stiffness matrix increases.

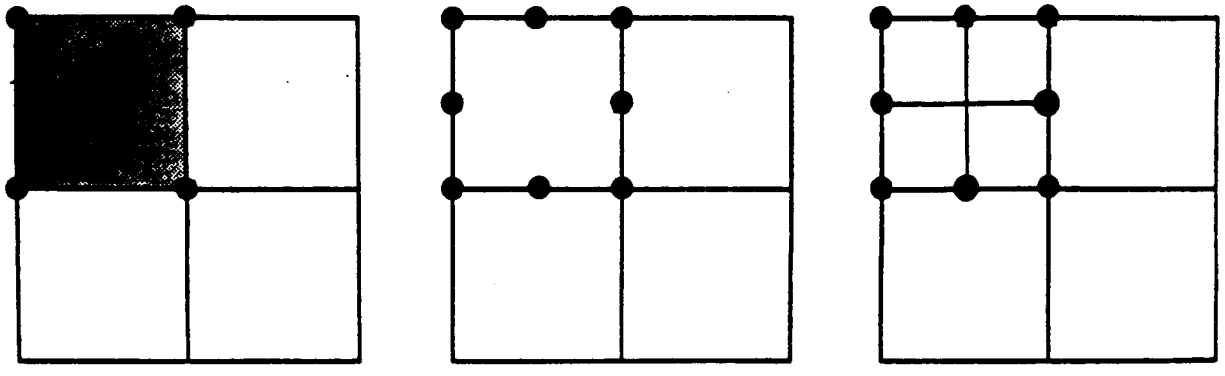


Fig. 16 Schemes for mesh refinement.

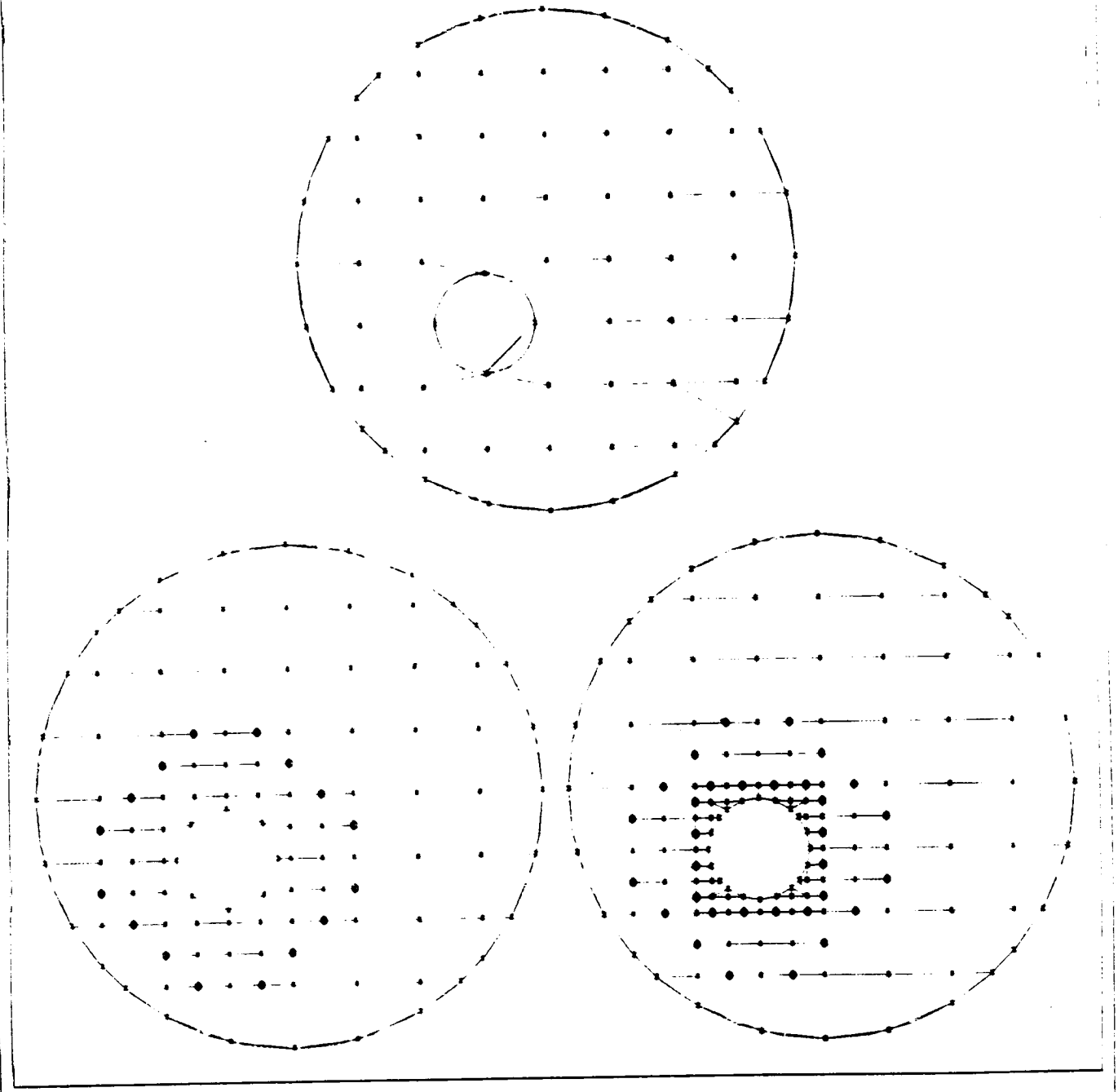


Fig. 17 Two stages of h-refinement.

In h-refinement existing elements are subdivided into smaller elements of the same type, as in Figure 16(c). To improve the geometric accuracy, localized h-refinement is done on the original geometric model rather than on the current finite element approximation. Thus, to refine a particular element, one deletes the element, creates and classifies new vertices and nodes, and inserts the smaller new elements into the grid. Discontinuities of displacements along edges where smaller elements abut on larger elements are avoided by using constraint equations. These are indicated by the circled nodes in Figure 16(c).

Figure 17 shows examples of localized refinement. Note that successive h-refinements improve the geometric approximation of the original solid. A maximum cross element grading ratio of 2:1 is maintained during refinement.

Storage for the new entities created by h-refinement could be provided by adding a whole new bottom layer to the grid, but this would be wasteful unless very extensive h-refinement is needed. If the h-refinements are sparse, small localized explicit schemes or linked-list methods are more efficient.

Now assume that the original mesh has been refined in a few regions using the methods just described, that the affected elements have been tagged, and that the refined mesh is to be reanalyzed. Clearly one wants to do incremental analysis, i.e., to use partial results from the earlier analysis as much as possible. These results are available through the hierarchical grid, for example, using a tree of K

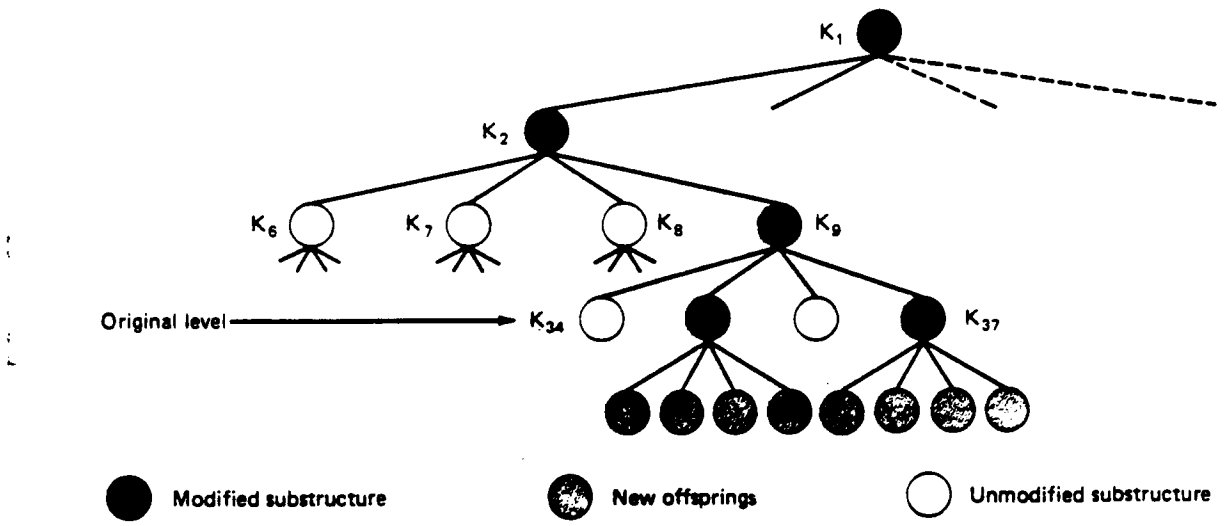


Fig. 18 Incremental reassembly.

matrices as in Figures 11 and 18.

The incremental FE assembler (Figure 1) traverses the tree and by examining the sons of each parent node, detects new offspring and computes the appropriate stiffness matrices (Figure 18). Stiffnesses for unmodified elements are recovered from storage, and new and old stiffnesses are combined to form a modified substructure. If a node has no new offspring, the complete old substructure is reused. The incremental solver (Figure 1) works similarly, inspecting tags on data to distinguish valid and invalid old results and reusing the former whenever possible.

Self-adaptive algorithm. Our current algorithm for controlling self-adaptive incremental analysis operates as follows (see Figure 10). After a mesh (either initial or refined) has been analyzed, error indicators are computed for each element together with an estimate of the global error. If the global error exceeds a specified limit, the system calls for refinement and reanalysis in regions having large local errors. This process continues automatically until the global error estimate falls below the specified limit. This rather simplistic control strategy seems to work in the cases we have tested, but it is crude and some needed improvements will be noted.

Considerable research has been conducted on the sources and nature of errors in FE analysis, and on their relationship to mesh refinement schemes [3—7]. Research pertinent to p-refinement has yielded significant results, whereas results on h-refinement have been based mainly on 1-D studies and are fairly primitive.

Thus far we have done little research on errors and our current error measures are crude. As in [5], our element error indicator (ϵ_i) is merely the average of the stress jumps (J_s, normal and tangential) across each element's edges with dimension (h) and assuming linear isoparametric elements:

$$\epsilon_i^2 = \frac{1-\nu}{E}\,\frac{h}{24}\int_{\tau_i} J_s^2 d\tau$$

normalized by the strain energy of the displaced model. Our global error estimator is simply the sum of the element error indicators. Figure 19 shows the computed values of the element error indicators for a sample problem (a plate with a hole under traction). Note that, in the vicinity of the hole, the data imply high stress gradients because the error indicators are high. Figure 19(b) shows an automatic refinement resulting from this set of error indicators.

An improvement of the current algorithm would be to replace the single global error indicator, which now serves as a simple refine/don't refine switch, with a hierarchical series of regional error indicators. These can be computed bottom-up in the tree, and should force selective refinement in cases where the overall average error is small but errors in small regions are high. Additional improvements can be expected as more is learned about the nature of errors in FE analysis. Such research should also generate the information needed to study the convergence properties of self-adaptive schemes.

Advantages and Disadvantages

The main advantage of our approach is that mesh generation and mesh analysis are integrated and in effect collaborate under the control of the error evaluator. Thus, the mesher only refines regions where refinement is needed, and the analyzer only computes "what's new" about a refined mesh. This type of efficient adaptive behavior is, in our opinion, the key to efficient automatic FE analysis.

Some can argue that mesh generation and mesh analysis should not be integrated because integration precludes "mixing and matching", i.e. being able to analyze, through simple interface translators, a mesh from "any" CAD system or preprocessor using "any" popular analysis package. We believe that by the 1990s, however, the benefits of integration will outweigh those of mixing and matching.

Spatially localized substructuring is the driving principle in both the mesh generator and mesh analyzer. This principle derives from recursive spatial subdivision and is manifested in our hierarchical grid and its underlying tree. The tree

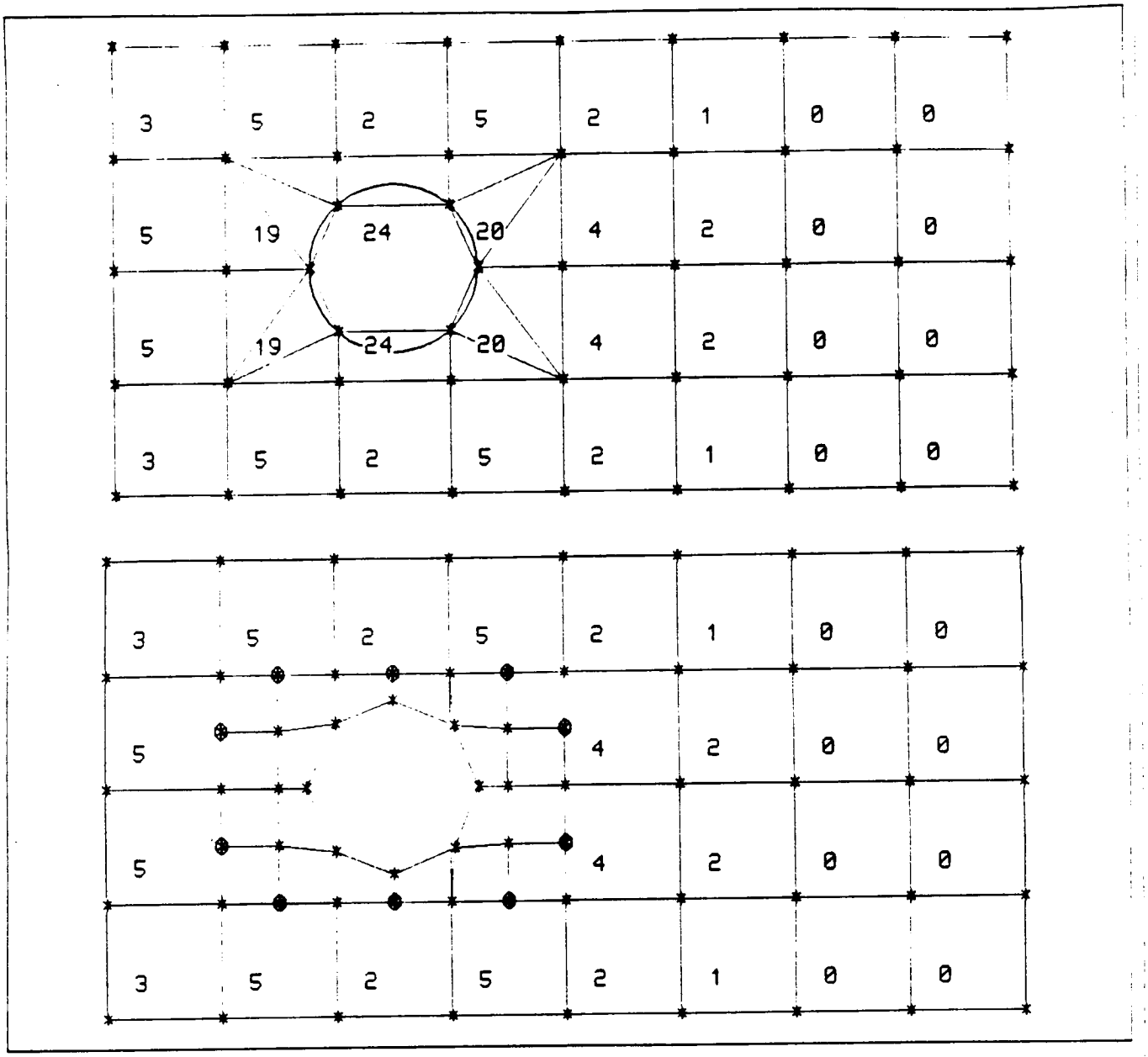


Fig. 19 Refinement driven by error indicator.

might be viewed as a generalization of the structure described in [19]. However, the latter is applied in subdomains that are mapped to regular figures (squares and triangles), and Rheinboldt's tree addresses the element partitioning induced in the regular figures. By avoiding mapping we are able to use the same structure for both meshing and analysis; further, the regularity of our structure permits systematic cell numbering and, hence, data access through calculated addresses rather than through searching or looking in tables.

This "divide-and-conquer" principle enables hard problems (such as object decomposition and equation-set solution) to be decomposed into smaller, tractable problems via spatial partitioning. We note that spatially localized substructuring, and spatial addressability in general, provide powerful mechanisms for coupling FE methods and results to other applications (e.g., manufacturing process modeling) through master data bases based on solid modeling.

Certain technical details already described, such as the regularity of the interior mesh elements, are also advantages of this approach.

Limitations. The main limitation of spatial subdivision methods is that they produce meshes that are dependent on orientation and position if the initial enclosing box is not tight.

This is most easily seen in simple objects that have a single, natural orientation. As such objects are rotated in a fixed set of subdivision axes the induced meshes change, often dramatically. Figure 20 is an example with a simple object meshed in a nonstandard orientation. Skilled analysts call such meshes "unnatural," and note that they usually contain more elements than "hand-made" meshes.

Spatial subdivision can be applied in non-Cartesian domains. For example, predominantly circular 2-D objects can be meshed efficiently in polar coordinates by subdivision of (r,θ). The meshes so produced can be managed through the

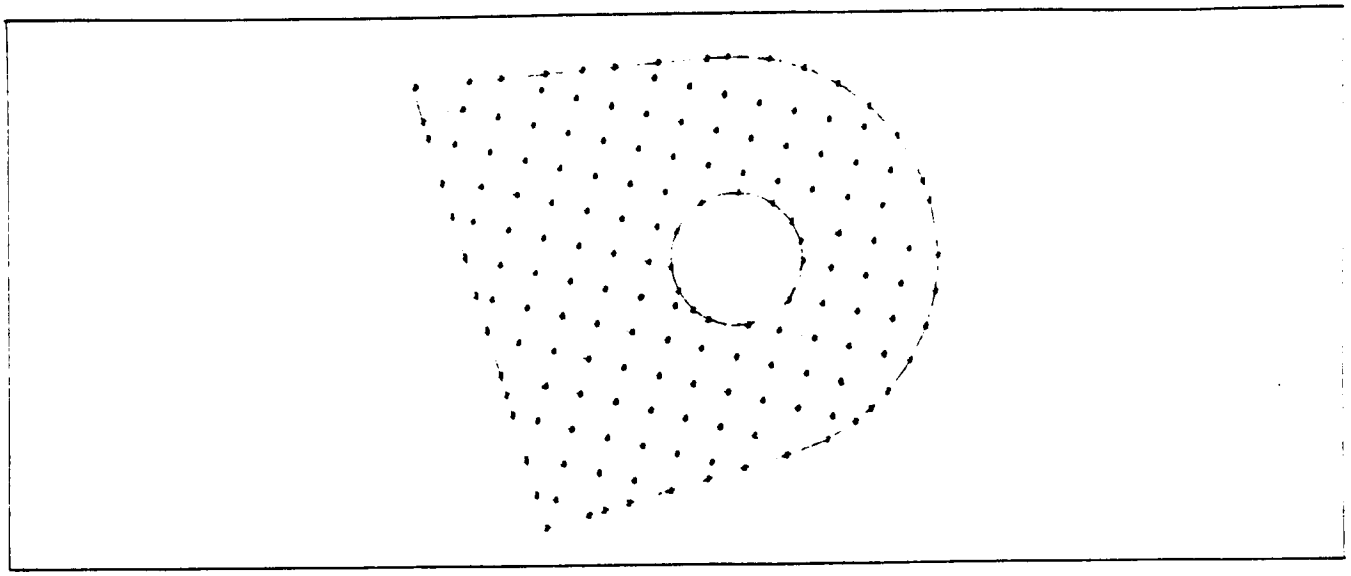


Fig. 20 Orientation and position dependence of meshes derived by spatial subdivision.

same hierarchical grid as is used for Cartesian subdivision [20]. Various schemes have been proposed for mixing subdivision strategies to cater to objects having both circular and rectilinear regions, but none seem promising [20].

The essential counter arguments are that "unnatural" meshes will produce valid results if the elements are valid, and that these results should converge under adaptive remeshing and reanalysis to a single set of (correct) results that is independent of position and orientation. Experimental evidence indicates that our approach exhibits such qualities.

Still To Be Resolved

Over the long term, four areas will require extensive theoretical work to make truly automatic FE analysis possible:

- *Error measures and indicators*. Better measures than the ones we use currently are needed, but they need not be optimal if adaptive convergence can be guaranteed.
- *Adaptive convergence*. We have seen no experimental evidence of divergence in the self-adaptive process, but automatic analysis systems like ours will require human monitoring to guard against divergence until strong convergence properties can be guaranteed.
- *Computational complexity*. We think that spatial substructuring techniques are asymptotically more efficient than the methods used in current solvers, but we have no results to prove or disprove this. Complexity and convergence analyses, when coupled, should provide bounds on the inherent cost of finite element analysis.
- *Nonlinear analysis*. Thus far we have confined our efforts to linear analysis but our approach to substructuring appears promising for nonlinear analysis as well.

Two other issues are currently more pressing: extending the systems to 3-D problems and handling loads and constraints automatically.

We have done 3-D work in parallel with our 2-D work. An efficient publicly available interior mesher (octree generator) has been created for solids describable in the PADL-2 solid modeling system [21, 22]. Figure 21 shows an example. The 2-D spatial substructuring techniques for managing analysis, adaptive remeshing, and reanalysis extend gracefully to 3-D, and indeed most of the 2-D control code is directly usable in 3-D. The major unresolved problems are in stage 2 of the automatic meshing procedure, i.e., in the handling of NIO cells. Promising methods for resolving these problems are being studied.

The handling of loads and constraints is the only aspect of 2-D linear FE analysis that we have not yet automated. At present, loads and constraints are applied manually when the assembler has completed its initial pass and the solver is about to begin its initial pass, i.e., at the transition between Figures 12(d) and 13(a). This raises two different questions.

First, there are no fundamental barriers to automating the application of loads and constraints at this stage of the solution procedure. The problems are strictly of an engineering nature. Essentially, what mechanisms should be provided in a solid modeler to support the declaration of loads and constraints (see Figure 1), and how should declarations be translated into mesh-node vector values? The translation problem is straightforward given a good solution to the declaration problem, and an experimental system with enough power to handle load and constraint declarations is already running under 3-D PADL-2 [23].

The second question is deeper. Should loads and constraints be applied at the outset, where they will influence construction of the initial mesh, rather than after an initial mesh has been built? This is certainly the case when meshes are constructed manually, and part of the analyst's skill is in knowing how fine a mesh should be in a loaded or constrained region. Should our mesher be modified to mimic this skill? The only possible gain we see is efficiency and this might be marginal because the current system already refines meshes automatically to reflect loads and constraints but only after it has passed from initial mesh analysis to adaptive remeshing and reanalysis.

In conclusion, we believe that the experimental system

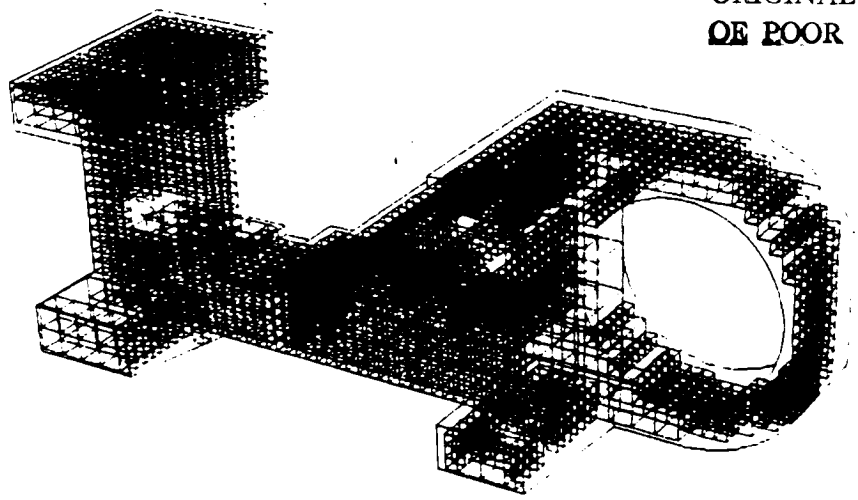


Fig. 21 Automatically derived octree decomposition of "Gehause" (a standard benchmark part for solid modeling systems). Here only the IN octree cells are displayed.

described here and its underlying principles represent a milestone on the road to truly automatic finite element analysis. ■

Acknowledgments

John Goldak of Carleton University contributed to this research and to the education of its authors. Victor Genberg of Eastman Kodak Company provided advice and encouragement. The plots were produced on equipment donated by Tektronix, Inc. Other industrial associate companies of the Production Automation Project provided both equipment and funds. Sustaining support was provided by the National Science Foundation under grants ECS-8104646 and DMC-8403882. The findings and opinions expressed here do not reflect the views of the sponsors.

References

1 Requicha, A.A.G. and Voelcker, H.B., "Solid Modeling: A Historical Summary and Contemporary Assessment," *IEEE Computer Graphics and Applications*, Vol. 2, No. 2, pp. 9–24, March 1982.

2 Requicha, A.A.G. and Voelcker, H.B., "Solid Modeling: Current Status and Research Directions," *IEEE Computer Graphics and Applications*, Vol. 3, No. 7, pp. 25–37, Oct. 1983.

3 Babuska, I. and Rheinboldt, W.C., "A-posterior Error Estimates for the Finite Element Method," *International Journal For Numerical Methods In Engineering*, Vol. 112, pp. 1597–1615, 1978.

4 Peano, A.G., Pasini, A., Riccioni, R., and Sardella, L., "Adaptive Approximation in Finite Element Structural Analysis," *Computer and Structures*, Vol. 10, pp. 332–342, 1979.

5 Kelly, D.W., Gago, J.P., Zienkiewicz, O.C., and Babuska, I., "A Posteriori Error Analysis and Adaptive Processes in the Finite Element Method: Part I. Error Analysis," *International Journal For Numerical Methods In Engineering*, Vol. 19, pp. 1593–1619, 1983.

6 Gago, J.P., Kelly, D.W., Zienkiewicz, O.C. and Babuska, I., "A Posteriori Error Analysis and Adaptive Processes in the Finite Element Method: Part II. Adaptive Mesh Refinement," *International Journal For Numerical Methods In Engineering*, Vol. 19, pp. 1621–1656, 1983.

7 Zienkiewicz, O.C., Gago, J.P., and Kelly, D.W., "The Hierarchical Concept in Finite Element Analysis," *Computers and Structures*, Vol. 16, No. 1–4, pp. 53–65, 1983.

8 Kela, A., "Automatic Finite Element Mesh Generation and Self-Adaptive Incremental Analysis Through Solid Modeling," Dissertation, Production Automation Project, University of Rochester, 1986 (in preparation).

9 Wordenweber, B., "Finite Element Mesh Generation," *Computer-Aided Design*, Vol. 16, No. 5, pp. 285–291, Sept. 1984.

10 Cavendish, J.C., Field, D.A., and Frey, W.H., "An Approach to Automatic Three-Dimensional Finite Element Mesh Generation," *International Journal For Numerical Methods In Engineering*, Vol. 21, pp. 329–347.

11 Lee, Y.T., "Automatic Finite Element Mesh Generation Based On Constructive Solid Geometry," Dissertation, Mechanical Engineering Dept., University of Leeds, England, April 1983.

12 Jackins, C.L. and Tanimoto, S. L., "Octrees and Their Use in Representing Three-Dimensional Objects," *Computer Graphics and Image Processing*, Vol. 4, No. 3, pp. 249–270, Nov. 1980.

13 Yerry, M. A. and Shephard, M. S., "A Modified Quadtree Approach to Finite Element Mesh Generation," *IEEE Computer Graphics and Applications*, Vol. 3, No. 1, pp. 39–46, Jan./Feb. 1983.

14 Yerry, M. A. and Shephard, M. S., "Automatic Three-Dimensional Mesh Generation by the Modified Octree Technique," *International Journal For Numerical Methods In Engineering*, Vol. 20, pp. 1965–1990, 1984.

15 Lee, Y.T. and Requicha, A.A.G., "Algorithms for Computing the Volume and Other Integral Properties of Solids: Part II. A Family of Algorithms Based On Representation Conversion and Cellular Approximation," *Communications of the ACM*, Vol. 25, No. 9, pp. 642–650, Sept. 1982.

16 Requicha, A.A.G., "Representations for Rigid Solids: Theory, Methods, and Systems," *ACM Computing Surveys*, Vol. 12, No. 4, Dec. 1980.

17 Requicha, A.A.G. and Voelcker, H.B., "Boolean Operations in Solid Modeling: Boundary Evaluation and Merging Algorithms," *Proceedings of the IEEE*, Vol. 73, No. 1, pp. 30–44, Jan. 1985.

18 Dodds Jr., R. H. and Lopez, L.A., "Substructuring in Linear and Nonlinear Analysis," *International Journal For Numerical Methods In Engineering*, Vol. 15, pp. 583–597, 1980.

19 Rheinboldt, W.O. and Mesztenyi, C.K., "On a Data Structure for Adaptive Finite Element Mesh Refinements," *ACM Transactions On Mathematical Software*, Vol. 6, No. 2, pp. 166–187, June 1980.

20 Kela, A., "Approaches to Automatic Finite Element Mesh Generation From CSG Representations of Solids," *ITM No. 43*, Production Automation Project, University of Rochester, July 1983.

21 Hartquist, E. E., "Public PADL-2," *IEEE Computer Graphics and Applications*, Vol. 3, No. 7, pp. 30–31, Oct. 1983.

22 Kela, A., "Programmer's Guide to the PADL-2 Octree Processor Output System," *Input/Output Group Memo No. 15*, Production Automation Project, University of Rochester, Jan. 1984.

23 Requicha, A. A. G. and Chan, S. C., "Representation of Geometric Features, Tolerances, and Attributes in Solid Modelers Based On Constructive Geometry," *ITM No. 48*, Production Automation Project, University of Rochester, Oct. 1985.